Boolean Exponent Splitting

Michael Tunstall¹, Louiza Papachristodoulou² and Kostas Papagiannopoulos³

¹Rambus, 4453 North First Street, Suite 100, San Jose, California, U.S.A.
²Fontys University of Applied Sciences, Rachelsmolen 1, Eindhoven, The Netherlands
³University of Amsterdam, SNE-CCI, Science Park 904, Amsterdam, The Netherlands

Keywords: Exponent Splitting, Side-channel Attacks, Countermeasures.

Abstract: A typical countermeasure against side-channel attacks consists of masking intermediate values with a random number. In symmetric cryptographic algorithms, Boolean shares of the secret are typically used, whereas in asymmetric algorithms the secret exponent is typically masked using algebraic properties. This paper presents a new exponent splitting technique with minimal impact on performance based on Boolean shares, typically requiring only an extra register and a few register copies per bit. We perform a security evaluation of our algorithms using a mutual information framework and provide proofs that they are secure against first-order side-channel attacks. The side-channel resistance of the proposed algorithms are also practically verified with test vector leakage assessment performed on Xilinx’s Zynq zc702 evaluation board.

1 INTRODUCTION

Side-channel analysis as a method of extracting cryptographic keys was first presented by Kocher (Kocher, 1996), who noted that timing differences in the execution time of a modular exponentiation could be used to break instances of RSA (Rivest et al., 1978). Subsequently, Kocher et al. (Kocher et al., 1999) observed that the instantaneous power consumption could reveal information on intermediate states of any cryptographic algorithm, since the instantaneous power consumption has, in many cases, been shown to be proportional to the Hamming weight of the data being manipulated (Brier et al., 2004), and it was later shown that the electromagnetic emanations around a device can be exploited in the same way (Gandolfi et al., 2001; Quisquater and Samyde, 2001).

In public key cryptography, one typically uses countermeasures based on redundant representations to prevent side-channel leakage (Coron, 1999; Win et al., 1998) (referred to as blinding). To protect an exponent used in a group exponentiation one would typically add a random multiple of the order of the group to the exponent, providing a random bitwise representation of the exponent. These countermeasures can provide a strong resistance to Differential Power Analysis (DPA), but are not convenient in some instances. As noted by Smart et al. (Smart et al., 2008), the random value used to blind an exponent needs to have a bit length larger than the longest run of zeros or ones in the bitwise representation of the order of the group. If we consider ECDSA (National Institute of Standards and Technology (NIST), 2009), for example, the bitwise representations of the orders of the groups used contain long runs of ones making this countermeasure costly.

In this paper, we present a new countermeasure for exponent splitting. We describe a method of splitting an exponent into two Boolean shares, analogous to the countermeasures that one would use for an implementation of a block cipher and similar to the countermeasures used to prevent address-bit side-channel attacks (Messerges et al., 1999; Messerges and Dabby, 1999; Itoh et al., 2002). Having embedded devices as our targeted implementation, and an adversary able to get useful information from the length of the exponent or the intermediate values, we provide a number of secure algorithms against a broad range of side-channel attacks.

At the same time, the modifications that are required to a group exponentiation algorithm have negligible effect on the time required to compute the actual group exponentiation, which is a significant advantage over previous examples of exponent splitting (Clavier and Joye, 2001; Ciet and Joye, 2003). In
addition, our method can be efficiently combined with
blinding techniques applied to the input to a group ex-
pONENTation algorithm, in order to prevent leakage of
the intermediate values.

We present an evaluation of the method of Boolean
exponent splitting using the information-theoretic
framework of Standaert et al. (Standaert et al., 2009) and a Test Vector Leakage Assessment
(TVLA) by Goodwill et al. (Goodwill et al., 2011).
We investigate the usual leakage models based on data
or location leakage and show that an adversary would
need either a second-order data attack or a third-order
location attack to successfully break the security of
our algorithms. In addition, we present for the first
time a hybrid model, where data leakage is combined
with location leakage, offering new exploitation
opportunities. The rich interactions between data and
location leakage corroborates the need for holistic coun-
termeasures that encompass a wide spectrum of side-
channel attacks.

2 EXPONENT SPLITTING
METHODS

The critical operation in public key cryptographic al-
go
gorithms is exponentiation in a certain group \(G\) of or-
der \(\mu\), where the input message \(x \in G\) is raised by a
secret exponent \(\kappa\) and the result \(y = x^\kappa\) is the pub-
lic output of the algorithm. When implementing a
group exponentiation algorithm the exponent is typi-
cally blinded by adding some random multiple of
the order of the group to the exponent. Trivially,
\((r\mu) + \kappa \equiv \kappa \pmod{\mu}\) for \(r, \kappa \in \mathbb{Z}\) where \(r\) is random.
Hence, computing \(x^{r\mu + \kappa}\) is equivalent to computing
\(x^\kappa\). While this randomizes the bitwise represen-
tation of an exponent, the entire exponent is still equiva-
lent to the exponent in a given group. Examples of attacks
that have been proposed include analyzing a single
trace (from SPA (Kocher et al., 1999)) to collisions in
manipulated values (Witteman et al., 2011; Kim
et al., 2010; Hanley et al., 2015)) or attempting to find
collisions in the random values used to then derive a
(blinded) exponent (Schindler and Itoh, 2011).

One method that can hinder these attacks, is to
split an exponent into two values whose bitwise repre-
sentations are random. Then one would compute
a group exponentiation where the combined effect of
the two values is equivalent to that of the desired ex-
ponent. Randomly splitting the value that manipu-
lates secret data was proposed initially by Chari et.
\(\mu\) in (Chari et al., 1999) as a generic technique to
provide provable resistant implementation to side-
channel attacks. By randomly splitting every bit of
the original computation into \(m\) shares, where each
share is equiprobably distributed and every proper
subset of \(m-1\) shares is statistically independent of
the encoded bit, the cryptographic computation can
then be performed securely by computing only the
shares, without ever reconstructing the original bit.
The leakage from every computation does not reveal
any useful information to the adversary, who needs
to perform \(m\) attacks to reconstruct the secret. There
are several methods of exponent splitting proposed by
Clavier and Joye (Clavier and Joye, 2001):

- **Additive Splitting.** For a random integer \(r\) with
  bit-length smaller or equal to the exponent \(\kappa\), we
can define \(\kappa = r + (\kappa - r)\). The output of the mod-
ular exponentiation \(y = x^\kappa\) in \(G\) can be computed by
\(y = x^r \cdot x^{\kappa - r}\) in \(G\).
- **Multiplicative Splitting.** For some group \(G\) we
can define \(k' = k \cdot r^{-1} \mod|G|\) for some integer \(r\).
Then the exponentiation \(y = x^k\) in \(G\) can be
computed by using \(y = (x^r)^{k'} \mod|G|\).

The same techniques can be applied to scalar multi-
plification algorithms for elliptic curves (ECS), in or-
der to hide the secret scalar. The problem with these
methods of exponent splitting is that one is required
to know the order of the group \(G\), which may not be
available in some instances. They will also typically
double the time required to compute a group ex-
pONENTiation, because \(r\) is required to have a bit-length
similar to the exponent. A practical attack by Feix
et al. (Feix et al., 2014) demonstrates that a blinded
scalar can be determined if \(r\) is too small.

A further method described by Ciet and Joye (Ciet
and Joye, 2003) is:

- **Euclidean Splitting.** By writing the exponent as
\(k = \lfloor k/r \rfloor r + k \pmod r\) and letting \(s = x^r\) for some \(r\),
then \(y = x^k\) can be computed by \(y = x^s \cdot x^{k \pmod r} =
(x^r)^{k'} \cdot x^k \pmod r\), where \(k' = \lfloor k/r \rfloor\).

The impact on the time required to compute an ex-
pONENTiation is lower than the other splitting methods
listed above. In fact, in (Ciet and Joye, 2003) the au-
thors evaluated this variant applied to Shamir’s dou-
ble ladder to have the same cost as the ‘double-and-
add-always’ algorithm (equivalent to the ‘square-and-
multiply-always’ for exponentiation). Precomputation
of powers of \(s\) can reduce the exponentiation cost
compared to additive or multiplicative splitting. How-
ever, this method has the same constraints as adding
a multiple of the group order. That is, \(r\) needs to have
a bit length larger than the longest run of ones and
zeros in \(k\) and may have a significant impact on per-
formance (Smart et al., 2008). A secure division al-
gorithm is also required, see, for example, Joye and
Villegas (Joye and Villegas, 2002).
3 BOOLEAN EXPONENT SPLITTING METHODS

In this section, we propose methods of exponent splitting based on XOR operation, and how an XOR-split exponent can be applied to the Montgomery powering ladder.

3.1 Montgomery Powering Ladder

The Montgomery Powering Ladder (MPL) was originally proposed as a means of speeding up scalar multiplication over ECs and later shown to be applicable to multiplicative written Abelian groups (Montgomery, 1987; Joye and Yen, 2002). We recall the description of the MPL given by Joye and Yen (Jaye and Yen, 2002): We consider the problem of computing \( y = x^k \) in \( \mathbb{G} \) for inputs \( x \) and \( k \). Let \( \sum_{i=0}^{n-1} k_i 2^i \) be the binary expansion of \( k \) with bit length \( n \) (for ease of expression we shall also denote this as \((k_n, \ldots, k_0)_2\) where convenient). Then, defining \( L_j = \sum_{i=j}^{n-1} k_i 2^i \) and \( H_j = L_j + 1 \), we have \( L_j = 2 L_{j+1} + k_j = L_{j+1} + H_{j+1} + k_j - 1 = 2 H_{j+1} + k_j - 2 \) and so we obtain

\[
\begin{align*}
(L_j, H_j) = \begin{cases} 
(2 k_{j+1} + H_{j+1}, H_{j+1}) & \text{if } k_j = 0, \\
(L_{j+1} + 2 H_{j+1}, 2 H_{j+1}) & \text{if } k_j = 1.
\end{cases}
\end{align*}
\]

(1)

If we consider one register containing \( x^{L_j} \) and another containing \( x^{H_j} \), then (1) implies that

\[
\begin{align*}
(x^{L_j}, x^{H_j}) = \begin{cases} 
( (x^{L_j+1})^2, x^{L_j+1}) & \text{if } k_j = 0, \\
(x^{L_j+1}, (x^{H_j+1})^2) & \text{if } k_j = 1.
\end{cases}
\end{align*}
\]

(2)

Given that \( L_0 = 0 = k \) one can build an exponentiation algorithm that requires two group operations per bit of the exponent. Joye and Yen give several different versions, one of which is shown in Algorithm 1. All these methods are highly regular, meaning that a deterministic sequence of operations is executed for an exponent of a given bit length.

Algorithm 1: Montgomery Ladder.

**Input:** \( x \in \mathbb{G} \), an \( n \)-bit integer \( k = \sum_{i=0}^{n-1} k_i 2^i \)

**Output:** \( x^k \)

1. \( R_0 \leftarrow 1_G ; R_1 \leftarrow x ; \)
2. for \( i = n \) down to 0 do
3. \( R_{k_i} \leftarrow R_{k_i} \cdot R \cdot x \):
4. \( R_{k_i} \leftarrow (R_{k_i})^2 ; \)
5. end
6. return \( R_0 \)

In applying an XOR-split exponent to MPL we use one share to dictate the address accessed and the other to act as the exponent. That is, we consider (1), where the previous round may provide either \((L_j, H_j)\) or \((H_j, L_j)\) and the computation changed accordingly.

Let \( S_{0,j} = L_j \) and \( S_{1,j} = H_j \) and \( \sum_{i=0}^{n-1} a_i 2^i \) be the binary expansion of \( A \) with bit length \( n \) (i.e., the same bit length as the exponent). Then we can use the values of \( a_i \) to dictate whether a pair of registers holds \((L_j, H_j)\) or \((H_j, L_j)\). Specifically, (1) can be rewritten as:

\[
\begin{align*}
(S_{a_j}, S_{\neg a_j}) = \begin{cases} 
(2 S_{a_{j+1}} + A_{a_{j+1}} + S_{\neg a_{j+1}}, S_{a_{j+1}}) & \text{if } k_j = 0, \\
(S_{a_{j+1}} + S_{\neg a_{j+1}}, 2 S_{a_{j+1}}) & \text{if } k_j = 1.
\end{cases}
\end{align*}
\]

(3)

In (3), the values of \( L_j \) and \( H_j \) are assigned to \( S \) in an order dictated by the binary expansion of \( A \). Generating \( A \) as a random sequence of bits could provide some side-channel resistance, but does not protect the exponent.

We further consider \( \sum_{i=0}^{n-1} a_i 2^i \) and \( \sum_{i=0}^{n-1} b_i 2^i \) be the binary expansion of \( A \) and \( B \), respectively, where \( \kappa = A \oplus B \) of bit length \( n \). We note that, as above, \( \sum_{i=0}^{n-1} b_i 2^i \) is the binary expansion of \( \kappa \) and \( k_i = a_i \oplus b_i \) for \( 0 \leq i < n \). Then (3) can be rewritten as:

\[
\begin{align*}
(S_{a_j}, S_{\neg a_j}) = \begin{cases} 
(2 S_{a_{j+1}} + 2 a_{j+1} + S_{\neg a_{j+1}}, S_{a_{j+1}}) & \text{if } k_j = 0, \\
(S_{a_{j+1}} + 2 a_{j+1}, 2 S_{a_{j+1}}) & \text{if } k_j = 1.
\end{cases}
\end{align*}
\]

(4)

Rather than using the same value to control which order \( L_j \) and \( H_j \) are assigned and read, we use the bits of \( \kappa \) to determine the order \( L_j \) and \( H_j \) are assigned, and the bits of \( B \) to determine the order they are read.

The combined effect is that the order \( L_j \) and \( H_j \) are assigned and read is dictated by the bits of \( \kappa \).

\[
\begin{align*}
(S_{a_j}, S_{\neg a_j}) = \begin{cases} 
((2 S_{a_{j+1}} + 2 a_{j+1})^2, S_{a_{j+1}}) & \text{if } k_j = 0, \\
(S_{a_{j+1}}, (2 S_{a_{j+1}} + 2 a_{j+1})^2) & \text{if } k_j = 1.
\end{cases}
\end{align*}
\]

(5)

From which we can define Algorithm 2, which operates in much the same way as the MPL, as it produces a regular sequence of multiplications and squaring operations. However, one more register is required to allow the assignment in line 5 to affect \( R_0 \) or \( R_1 \).

This algorithm is the basis that we use to present the essence of Boolean-split exponent. Algorithm 2 is largely equivalent to an algorithm proposed by Izumi et al. (Izumi et al., 2010) where we set the multiplication in line 4 to operate in a random order as it provides a better resistance to collision attacks, as demonstrated by Kim et al. (Kim et al., 2010). We discuss this further in Section 3.2.

The intermediate states of the registers are not randomized in Algorithm 2 and would require additional countermeasures to provide a secure implementation.
Algorithm 2: Montgomery Ladder with XOR-Split Exponent I.

Input: $x \in G$, $n$-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$ and $B = \sum_{i=0}^{n-1} b_i 2^i$

Output: $x^k$ where $k = A \oplus B$

1. $R_0 \leftarrow 1_G$ ; $R_1 \leftarrow 1_G$ ; $R_2 \leftarrow 1_G$
2. $b' \leftarrow \mathbb{Z}_2 \{0, 1\}$ ; $R_{a_0} \leftarrow x$
3. for $i = n - 1$ do down to 0 do
   4. $R_2 \leftarrow R_{a_0} \cdot R_{a_1}$
   5. $R_{a_0} \leftarrow (R_{[k \not\equiv b']} \oplus a_0)^2$
   6. $R_{a_1} \leftarrow R_2$
   7. $b' \leftarrow b_1$
8. end return $R_{b'}$

For example, inexpensive solutions such as randomizing projective points (Win et al., 1998) or Ebeid and Lambert’s blinding method for RSA (Ebeid and Lambert, 2010) can be used (see Section 5). If we assume that the values held in registers $\{R_0, R_1, R_2\}$ do not leak (i.e., we only consider whether the exponent leaks) we can state the following:

Lemma 1. Assuming that the values held in registers $\{R_0, R_1, R_2\}$ do not leak, an implementation of Algorithm 2 is resistant to first-order side-channel analysis.

Proof. It suffices to consider each intermediate state and verify that at least one random mask is applied. Verifying this for an entire group exponentiation would be tedious, but can be simplified if we consider two rounds of Algorithm 2. That is, if we consider round $m$, where $0 \leq m \leq n - 2$, then the following operations are performed:

1. $R_2 \leftarrow R_{a_0} \cdot R_{a_1}$
2. $\alpha \leftarrow b_m \oplus b'$
3. $\beta \leftarrow \alpha \oplus a_m$
4. $R_{a_0} \leftarrow R_2$
5. $R_{a_1} \leftarrow R_2$
6. $R_{a_0} \leftarrow R_{a_0} \cdot R_{a_1}$
7. $\alpha \leftarrow b_{m+1} \oplus b_m$
8. $\beta \leftarrow \alpha \oplus a_{m+1}$
9. $R_{a_1} \leftarrow R_2$
10. $R_{a_0} \leftarrow R_{a_0} \cdot R_{a_1}$

Let the proposition $P(n)$ be that round $n > 0$ is resistant to first-order side-channel analysis for the $n$-th treated bit of the exponent. If we consider the first round, we wish to show $P(1)$ is true and, in the above code fragment, $b'$ is set to a random value from $\{0, 1\}$. Then, it is easy to see that:

- the results of the operations in lines 2, 3, 7 and 8 are uniformly distributed on $\{0, 1\}$.

If we assume that $P(m)$ is true for all $m \in \{1, \ldots, n\}$, then we consider $P(n + 1)$ where $b'$ is set to $b_n$. As $b_n$ is one share of a previously treated exponent bit, it is indistinguishable from a random value from $\{0, 1\}$. The above statements regarding the results of the operations apply. Hence, by induction we have shown $P(n)$ is true for all $n > 0$. To complete the proof, we simply note that only half of the code fragment above will need to be considered in the last round. □

Remark. In (Itoh et al., 2003), the authors present the randomized addressing method (RA), in order to provide protection against address-based DPA and eliminate the correlation between an exponent bit and the register where the result of an operation is stored. In this work, we do not limit our countermeasure to work only against address-based DPA. Our goal is to perform operations on different exponent shares, in a way that an adversary would need a combination of leakages (such as higher-order DPA combined with template attacks) in order to recover the exponent.

3.2 Using Inverses

In this section we propose an algorithm more suited to groups where inversions can be readily computed. Le Duc et al. (Le et al., 2015) propose a straightforward variant of the Montgomery powering ladder that requires the computation of inverses. They note that (1) can be rewritten as:

$$P(J, H_j) = \begin{cases} (H_{j-1} + L_{j+1} + H_{j+1}) & \text{if } k_j = 0, \\ (H_{j+1} + L_{j+1} + L_{j+1}) & \text{if } k_j = 1. \end{cases}$$

From which we can define Algorithm 3. If we let $T_{0,j} = L_j$ and $T_{1,j} = H_j$, or $T_{0,j} = H_j$ and $T_{1,j} = L_j$ and store the ordering in another variable we can rewrite (6) as:

$$P(J, H_j) = \begin{cases} (H_{j-1} + L_{j+1} + H_{j+1}) & \text{if } k_j = 0, \\ (H_{j+1} + L_{j+1} + H_{j+1}) & \text{if } k_j = 1. \end{cases}$$

From which we can define Algorithm 4. Following the previous notation, we notice that $T_{0,j}$ should contain the sum of the registers in the previous round\(^1\). Therefore, (7) can be rewritten as follows:

$$P(J, H_j) = \begin{cases} (H_{j-1} + L_{j+1} + H_{j+1}) & \text{if } k_j = 0, \\ (H_{j+1} + L_{j+1} + H_{j+1}) & \text{if } k_j = 1. \end{cases}$$

\(^1\)The algorithms are left-to-right, so $j + 1$ indicates the round preceding $j$. 

324
Algorithm 3: Variant with Inverses I.

**Input:** $x \in \mathbb{G}$, an $n$-bit integer $\kappa = \sum_{i=0}^{n-1} k_i 2^i$

**Output:** $x^\kappa$

1. $R_0 \leftarrow 1_G$; $R_1 \leftarrow x$;
2. $U_0 \leftarrow x^{-1}$; $U_1 \leftarrow x$;
3. for $i = n - 1$ down to 0 do
   4. $R_{-i} \leftarrow R_{b_i} \cdot R_{-i}$;
   5. $R_{ki} \leftarrow R_{-i} \cdot U_{ki}$;
4. end
7. return $R_0$

Algorithm 4: Variant with Inverses II.

**Input:** $x \in \mathbb{G}$, an $n$-bit integer $\kappa = \sum_{i=0}^{n-1} k_i 2^i$

**Output:** $x^\kappa$

1. $R_0 \leftarrow 1_G$; $R_1 \leftarrow x$;
2. $U_0 \leftarrow x^{-1}$; $U_1 \leftarrow x$;
3. for $i = n - 1$ down to 0 do
   4. $R_{-i} \leftarrow R_{b_i} \cdot R_{-i}$;
   5. $R_{ki} \leftarrow R_{-i} \cdot U_{ki}$;
4. end
7. return $R_{d_0}$

We note that to treat $k_{j+1}$, $b' = k_j$. However, if we let $k_j = a_j \oplus b_j$, for $a_j, b_j \in \{0,1\}$ and $h = a_j \oplus b_j \oplus b_{j-1}$, we can modify (8) as follows:

$$ (b_0, b_h) = \begin{cases} (\alpha + 1, \beta + 1) & \text{if } a_h = b_h, \\ (\alpha, \beta) & \text{if } a_h = \neg b_h. \end{cases} $$

(9)

By using the above equations as exponents of $x$, we can define Algorithm 5.

Algorithm 5: Montgomery Ladder with XOR-Split Exponent II.

**Input:** $x \in \mathbb{G}$, $n$-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$ and $B = \sum_{i=0}^{n-1} b_i 2^i$, $r \in \mathbb{R} Z$

**Output:** $x^\kappa$ where $\kappa = A \oplus B$

1. $R_0 \leftarrow 1_G$; $R_1 \leftarrow 1_G$; $U_0 \leftarrow x$; $U_1 \leftarrow x^{-1}$;
2. $b' \leftarrow \{0,1\}$; $R_{-b'} \leftarrow x$;
3. for $i = n - 1$ down to 0 do
   4. $R_0 \leftarrow R_{b_i} \oplus R_{-b_i} \cdot R_{(b_i \oplus b')(\oplus a_i)}$;
   5. $R_1 \leftarrow R_0 \cdot U_{b_i}$;
   6. $b' \leftarrow b_i$;
4. end
7. return $R_{b'}$

Algorithm 5 follows the same sequence of instructions with the MPL. Its correctness can be verified by the fact that at every round the difference $R_0/R_1 = x$ or $R_0/R_1 = x$, as for the usual ladder step. The advantage of Algorithm 5 compared to Algorithm 2, and consequently previously proposed algorithms by Izumi et al. (Izumi et al., 2010), is the elimination of the auxiliary register $R_2$. Instead, the auxiliary registers $U_0, U_1$ manipulate the known fixed value $x$ or $x^{-1}$ for computational purposes, and they do not require additional computational power or updates when the algorithm is executed.

As previously, if we assume that the values held in registers $\{R_0, R_1\}$ do not leak we can state the following:

**Lemma 2.** Assuming that the values held in registers $\{R_0, R_1\}$ do not leak, an implementation of Algorithm 5 is resistant to first-order side-channel analysis.

**Proof.** It suffices to consider each intermediate state and verify that at least one random mask is applied. Verifying this for an entire group exponentiation would be tedious, but can be simplified if we consider two rounds of Algorithm 5. That is, if we consider round $m$, where $0 \leq m \leq n - 2$, then the following operations are performed:

1. $\alpha \leftarrow b_m \oplus b'$
2. $\beta \leftarrow a_m$
3. $\alpha \leftarrow b_m \oplus b_m$
4. $R_1 \leftarrow R_0 \cdot U_{b_m}$
5. $R_0 \leftarrow R_1 \cdot R_{b_m}$
6. $R_1 \leftarrow R_0 \cdot U_{b_m}$
7. $R_0 \leftarrow R_0 \cdot R_{b_m}$
8. $R_1 \leftarrow R_0 \cdot U_{b_m}$

Let the proposition $\mathcal{P}(n)$ be that round $n > 0$ is resistant to first-order side-channel analysis for the $n$-th treated bit of the exponent. If consider the first round, we wish to show $\mathcal{P}(1)$ is true and, in the above code fragment, $b'$ is set to a random value from $\{0,1\}$. Then, it is easy to see that:

- the results of the operations in lines 3, 4, 7 and 8 are dependent on the random values $\{R_0, R_1\}$.
- the results of the operations in lines 1, 2, 5 and 6 are uniformly distributed on $\{0,1\}$.

If we assume that all $\mathcal{P}(m)$ is true for $m \in \{1, \ldots, n\}$, then we consider $\mathcal{P}(n+1)$ where $b'$ is set to $b_n$. As $b_n$ is one share of a previously treated exponent bit, it is indistinguishable from a random value from $\{0,1\}$. The above statements regarding the results of the operations apply. Hence, by induction we have shown $\mathcal{P}(n)$ is true for all $n > 0$. To complete the proof, we simply note that only half of the code fragment above will need to be considered in the last round. \qed
3.3 Boolean Scalar Splitting

In the above, we define group exponentiations applicable to any multiplicatively written group \( G \). However, specific groups may have particular characteristics that means the algorithms above are not suitable as described. In this section, we discuss the algorithms in the context of a group formed from the points on an elliptic curve (EC). We define the EC over a finite field \( \mathbb{F}_q \), for a large prime \( q \). \( \mathcal{E} \) consists of points \((x, y)\), with \( x, y \in \mathbb{F}_q \), that satisfy, for example, the short Weierstraß equation

\[
\mathcal{E} : y^2 = x^3 + ax + b
\]

with \( a, b \in \mathbb{F}_q \), and the point at infinity denoted \( \mathcal{O} \). The set \( \mathcal{E}(\mathbb{F}_q) \) is defined as \( \mathcal{E}(\mathbb{F}_q) = \{ (x, y) \in \mathcal{E} | x, y \in \mathbb{F}_q \} \cup \{ \mathcal{O} \} \), where \( \mathcal{E}(\mathbb{F}_q) \) forms an Abelian group under the chord-and-tangent rule and \( \mathcal{O} \) is the identity element. Alternative equations with different representations of a neutral element are also used in cryptographic algorithms, such as Edwards curves (Edwards, 2007; Bernstein and Lange, 2009) and Montgomery curves (Montgomery, 1987). The scalar multiplication of a given point is a group exponentiation in \( \mathcal{E} \) that uses elliptic curve arithmetic, i.e. addition between points or scalar multiplication \( [\kappa]P \) for some integer \( \kappa \) \( \in \mathbb{Z} \), and is an important part of many cryptographic algorithms.

The algorithms presented above cannot be securely implemented as described because of the neutral element. In the short Weierstraß example, the neutral element \( 1_G \) is represented in \( \mathcal{E} \) as the point at infinity \( \mathcal{O} \) and cannot be manipulated in a regular way. That is, one would typically be obliged to test for a numerical representation of \( \mathcal{O} \) and conduct a different operation if it is detected. In practice, one would implement the algorithm such that the most significant bit (assumed to be set to one) is already treated by the pre-processing. For example, Algorithm 2 can be implemented as shown in Algorithm 6, and Algorithm 5 as shown in Algorithm 7.

As previously, if we assume that the values held in registers \( \{R_0, R_1, R_2\} \) do not leak we can state the following:

**Corollary 1.** Lemma 1 implies that an implementation of Algorithm 6 is resistant to first-order side-channel analysis.

**Corollary 2.** Lemma 2 implies that an implementation of Algorithm 7 is resistant to first-order side-channel analysis.

The exponent splitting methods detailed in this paper do not modify the intermediate states generated and one would expect that randomizing projective points would be adequate to provide a secure solution (Win et al., 1998). However, such multiplicative masking can be problematic if an attacker can choose and input that could produce a point with a coordinate set to zero, which cannot be blinded using a multiplication (Goubin, 2003). Hence, one would need to combine our algorithms with Coron’s countermeasures (Coron, 1999) and add a small multiple of the order of the group to the private key before it is used. The bit length of the multiplier needs to be chosen such that an attacker cannot predict the location of a zero-coordinate with sufficient reliability to make it visible in a side-channel attack. Having a 16-bit multiplier may be sufficient, depending on the signal-to-noise ratio of the platform. The advantage of combining these countermeasures is that one does not need to consider the longest runs of ones or zeros in the order of the group.
4 SECURITY EVALUATION

In this section, we discuss the security of the algorithms presented previously, first by making a comparison with the state-of-the-art algorithms and then by providing a security evaluation of Algorithm 2, proposed in this paper.

4.1 State-of-the-Art Comparison

In this section, we compare our proposed algorithms with a selection of algorithms discussed in the previous sections and summarize our observations in Table 1.

The first block of algorithms in Table 1 contains exponentiation algorithms using the Montgomery power ladder without splitting the exponent (Algorithm 1), with additive splitting or with variations of XOR-splitting (Algorithms 2, 3, 4). Multiplicative or Euclidean splitting are not included in this table, because in terms of security they have the same side-channel resistance as an algorithm with additive splitting. In terms of performance, the number of operations is the similar, unless the values $s^k$ are precomputed and stored in memory.

The second block of algorithms summarizes the behavior of the corresponding scalar multiplication algorithms. Algorithms 8 and 9 are presented in Section 5.

We note that none of the algorithms in their current form can prevent leakage from observing the intermediate values. However, intermediate values can be blinded with a random value as previously described.

4.2 Mutual Information-based Security Evaluation

Having established that the proposed exponent splitting algorithms are probing-secure against first-order side-channel attacks, we proceed to analyze the noise amplification stage of the proposed countermeasure. Analytically, we perform an evaluation of Boolean exponent splitting (as described by Algorithm 2) using the information-theoretic framework of Standaert et al. (Standaert et al., 2009). Analogous approaches can be conducted for all exponent splitting algorithms, yielding very similar results. Our analysis considers two sources of leakage, namely data-based leakage and location-based leakage (also known as address leakage). Using these two leakage sources, we demonstrate three possible attack paths against Algorithm 5, covering all possible combinations between leakage sources. Thus we show the noise amplification stage when only data-based leakage is exploited (data attack), when only location-based leakage is exploited (location attack) and finally the noise amplification stage when the adversary combines data and location leakage (hybrid attack).

4.2.1 Notation & MI Metric

In this subsection, random variables are denoted with capital letters. Instances of random variables and constant values are denoted with lowercase letters. Capital bold letters are used for random variable vectors and matrices and calligraphic font denotes sets. All simulations in this section are carried out with the identity leakage function. Observable data-based leakages of a certain intermediate value $v$ are denoted using subscript $L_v$. Likewise, observable location-based leakages caused by accessing register $R_i$ (where $i$ is the index) are denoted using subscript $L_{R_i}$. To distinguish between data-based leakage and location-based leakage we use superscripts $L^\text{data}$ and $L^\text{loc}$. In addition, we assume that different sources of leakage (data, location) have different noise levels i.e. we assume homoscedastic data noise $N^\text{data} \sim \mathcal{N}(0, \sigma^2_{\text{data}})$ and homoscedastic location noise $N^\text{loc} \sim \mathcal{N}(0, \sigma^2_{\text{loc}})$.

We use the following formula to compute the MI metric:

$$MI(S; L) = \sum_{s \in S} \sum_{m \in M} \sum_{l \in L} \Pr[s|l] \log_2 \frac{\Pr[s|l]}{\Pr[s] \Pr[l]} \cdot d$$

where $d = \int_{L^{(d+1)}} \Pr[l|x,m] \cdot \log_2 \Pr[s|l] \cdot dl$ and

$$\Pr[s|l] = \frac{\sum_{m \in M} \Pr[s|m] \cdot \Pr[l]}{\sum_{m \in M} \Pr[l|m]}$$

and random variable $S$ denotes the secret exponent bit, $L$ denotes the leakage vector and $M$ is a d-dimensional randomness vector that we need to sum over when randomization is in place, i.e. $d$ is the attack order.

4.2.2 Data Leakage Attack

The first obvious way to recover $k_{n-1}$ is by observing the data leakage of the values $b_{n-1}$ and $d_{n-1}$ at the same time. We run the algorithm for the first two rounds and note the intermediate values that can leak information. We let $b'$ be a random value from $\mathbb{R}\{0,1\}$, then:
Table 1: Comparison Table.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Operations</th>
<th>Registers</th>
<th>Hide length</th>
<th>ADPA</th>
<th>Intern.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>( n \cdot M + n \cdot S )</td>
<td>2</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Clavier-Joye (Clavier and Joye, 2001)</td>
<td>( 2(n \cdot M + n \cdot S) )</td>
<td>2</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>( n \cdot M + n \cdot S )</td>
<td>3</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>Algorithms 3–4</td>
<td>( 2n \cdot M )</td>
<td>4</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>Algorithm 5</td>
<td>( 2n \cdot M )</td>
<td>4</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
</tr>
<tr>
<td>Algorithm 6</td>
<td>((n - 1) \cdot A + (n - 1) \cdot D)</td>
<td>3</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Algorithm 7</td>
<td>(2 \cdot (n - 1) \cdot A)</td>
<td>4</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Itoh et al. (Itoh et al., 2003) Alg. 8</td>
<td>((n - 1) \cdot D + (n - 1) \cdot A + 1 \cdot I)</td>
<td>3</td>
<td>( \times )</td>
<td>( \checkmark )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Izu et al. (Izu et al., 2010) Alg. 2</td>
<td>((n - 1) \cdot D + (n - 1) \cdot A)</td>
<td>3</td>
<td>( \times )</td>
<td>( \checkmark )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Algorithm 8</td>
<td>(n \cdot D + n \cdot A)</td>
<td>3</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Algorithm 9</td>
<td>(2 \cdot n \cdot A)</td>
<td>4</td>
<td>( \checkmark )</td>
<td>( \checkmark )</td>
<td>( \times )</td>
</tr>
</tbody>
</table>

As can be observed in above, the value \( b_{n-1} \) is accessed in the first iteration \((i = n - 1)\) three times, once when \( b_m \) is calculated (line 1), once implicitly for the index of \( U_{b_{n-1}} \) (line 4) and finally for \( b' \) (line 5). The value \( a_{n-1} \) is accessed once during the first iteration \((i = n - 1)\) and it is not used in the second iteration \((i = n - 2)\). We notice that the value \( b_{n-1} \) is used implicitly again in the second iteration, since it is equal to \( b' \). An attacker observing the power leakage of this algorithm should be able to probe at two different points in time, in order to observe both leakages \( L_{b_{n-1}}^{\text{data}} \) and \( L_{b_{n-1}}^{\text{data}} \) and eventually the i.e., we conclude that a second-order attack is possible for this scheme. Note also that the adversary with ability to conduct horizontal side-channel attacks (Battistello et al., 2016) could observe the leakage of \( b_{n-1} \) multiple times, average them by computing \( L_{b_{n-1}}^{\text{data}} = \frac{1}{s} \sum_{s=1}^{s} L_{b_{n-1}}^{\text{data}} \) in order to reduce the noise level and finally perform a second-order attack. The results of the MI evaluation are visible in Figure 1. As expected, the exponent splitting scheme performs noise amplification and has a different slope compared to an unprotected exponentiation (Algorithm 1). In addition, we observe the curve’s horizontal shift to the right caused by the horizontal exploitation of the available leakage, i.e., we can quantify the effect of multiple leaky points for \( b_{n-1} \).

4.2.3 Location Leakage Attack

Let us assume that the adversary can distinguish between the manipulation of registers according to which address is accessed, similar to the address-bit DPA attack described in (Izu et al., 2010). If the adversary can distinguish between accesses to, \( U_0 \) and \( U_1 \) for example, a direct consequence is recovery of value \( b_{n-1} \). To mount a successful attack against Algorithm 5 using solely location-based leakage, we need the simultaneous observation of the address of \( U_{b_{i-1}} \) and \( R_{a_i} \) for indexes \( i = b_{n-1} \) (line 4) and \( i = b_m \text{ or } a_m \) (line 3) and \( i = a_m \) (line 3). Thus, in order to recover \( b_{n-1} \), we need to observe leakage vector \( L_{b_{n-1}}^{\text{loc}} = [L_{b_{n-1}}^{\text{loc}}, L_{R_{a_i}}^{\text{loc}}, L_{R_{a_0}}^{\text{loc}}] \), i.e., perform a third-order attack. The results are visible in Figure 2, where we can observe the noise amplification effect that increases the curve’s slope. Naturally, a third-order attack using only location-based leakage tends to be less effective compared to a second-order attack using only data-based leakage. However, depending on the device, exploiting the address dependency may be more effective than exploiting the data dependency. That is, the third-order attack can become more efficient if \( \sigma_{\text{data}} > \sigma_{\text{loc}} \).

4.2.4 Hybrid Leakage Attack

Lastly, we analyze the scenario in which an adversary can observe both data-based and location-based leakage. Using this information the adversary can use leakage vector

\[
L = [L_{b_{n-1}}^{\text{data}} + L_{b_{n-1}}^{\text{loc}}]
\]

to carry out a second-order attack that uses data leakage to recover bit \( a_{n-1} \) and location leakage with regard to register \( U \) to recover bit \( b_{n-1} \). Since data and location leakage imply different noise levels, i.e., \( \sigma_{\text{data}} \neq \sigma_{\text{loc}} \), we need to represent the available information as a three-dimensional plot, as in Figure 3. The wave-like plot quantifies the attainable information with regard to a particular data and location noise level. Thus, it assists the side-channel evaluator to analyze the scheme’s security in a more holistic way that factors in location leakage and demonstrates the
tradeoff between data noise and location noise. If for instance $\sigma_{\text{loc}} \ll \sigma_{\text{data}}$ in the target device, the adversary can directly opt for the hybrid attack, instead of purging a data-only attack route.

\[ L = [L_{\text{data}}, L_{\text{loc}}] \]

Our implementations were developed using Xilinx’s Zynq xc702 evaluation board. The Zynq xc702 microprocessor contains two ARM7 cores and an FPGA fabric. We used one ARM7 core for our implementations, clocked at 667 MHz, and the FPGA provided a means of triggering an oscilloscope at a convenient point in our implementations. We acquired a trace of the electromagnetic emanations around one of the coupling capacitors.

The test that we used from TVLA is to determine whether there are statistically significant differences in the mean traces of two sets of traces, one acquired with a fixed scalar and the other with random scalar. One would typically randomly interleave acquisitions so that environmental effects are the same for both sets and there are no erroneous indications of leakage, caused, for example, by the least significant bit of a variable used to count the number of acquisitions. In applying this, one would take two sets of data, and conduct Welch’s $t$-test point-by-point to determine whether there is evidence against the null hypothesis that the sets are the same. We determine that leakage is present if we observe values above $6.63 \sigma$ which gives the probability of indicating leakage where no leakage is present, often referred to as a Type I error, of approximately $1 \times 10^{-5}$ when using traces containing $3 \times 10^5$ samples. The interested reader is referred to Goodwill et al. (Goodwill et al., 2011) and Schneider and Moradi (Schneider and Moradi, 2015) for a thorough description.

We made a straightforward implementation of Algorithm 6 using NIST’s P192 curve and conducted a test where we compared a set of traces with a fixed scalar compared to a set of traces with a random scalar. One would typically randomly interleave acquisitions so that environmental effects are the same for both sets and there are no erroneous indications of leakage, caused, for example, by the least significant bit of a variable used to count the number of acquisitions. In applying this, one would take two sets of data, and conduct Welch’s $t$-test point-by-point to determine whether there is evidence against the null hypothesis that the sets are the same. We determine that leakage is present if we observe values above $6.63 \sigma$ which gives the probability of indicating leakage where no leakage is present, often referred to as a Type I error, of approximately $1 \times 10^{-5}$ when using traces containing $3 \times 10^5$ samples. The interested reader is referred to Goodwill et al. (Goodwill et al., 2011) and Schneider and Moradi (Schneider and Moradi, 2015) for a thorough description.

We made a straightforward implementation of Algorithm 6 using NIST’s P192 curve and conducted a test where we compared a set of traces with a fixed scalar compared to a set of traces with a random scalar. The elliptic curve points were implemented as homogeneous projective points. We use the $x$ and $z$-coordinates in conjunction with so-called $x$-only al-

5 IMPLEMENTATION CONSIDERATIONS

In this section, we describe the results of applying Test Vector Leakage Assessment (TVLA) (Goodwill et al., 2011) to implementations of some of the algorithms above. We further describe modifications required to achieve a secure implementation where the hardware architecture can mean that variables that should be independent leak at the same time, potentially unmasking a secret value (Balasch et al., 2015).
gorithms for point arithmetic (Brier and Joye, 2002), as one would for an implementation of ECDH. The instantaneous electromagnetic emanations around the targeted capacitor were measured during the execution of the first 20 rounds of the implementation. The top-left trace in Figure 4 shows the result of a TVLA analysis with $1 \times 10^3$ traces where leakage can be seen in numerous places.

A straightforward implementation of Algorithm 6 was tested in the same way. The algorithm is similar to that proposed by Izumi et al. (Itoh et al., 2003) but with masking conducted before the execution of the scalar multiplication, rather than on-the-fly. The resulting TVLA traces is shown in the top-right of Figure 4, where we note that significant leakage is present with $1 \times 10^3$ traces. This is caused by the microprocessor combining values held in registers because of the architecture chosen by the designers (Balasch et al., 2015).

A more secure implementation can be made by computing some of the required indices before the execution of the main loop of the scalar multiplication, as shown in Algorithm 8. We set $C$ to $B \oplus \left[ \frac{2}{2} \right]$ such that individual bits of $B$ are masked by adjacent bits. The resulting TVLA trace is shown in the bottom-left of Figure 4, where we observe that there is only one place where we see significant leakage with $1 \times 10^6$ traces. This leakage occurs because the initial state of $\{R_0, R_1\}$ contain $\{P, 2P\}$ in some random order. In the first loop of the scalar multiplication $\{R_0, R_1\}$ is overwritten with $\{2P, 3P\}$ or $\{3P, 4P\}$, in some random order, depending on whether the second most-significant bit of $\kappa$ is set to 0 or 1, respectively. When $2P$ overwrites $2P$ the side-channel leakage will be significantly different to any other possible combination, since the Hamming distance will be zero.

A fully secure implementation can be achieved by randomizing the point produced by the doubling operation, by multiplying the $x$ and $z$-coordinate of the resulting point by a random value. In implementing Algorithm 8, this was achieved by randomizing $R_0$ and $R_1$ before the main loop of the scalar multiplication. The resulting TVLA trace is shown in the bottom-right of Figure 4, where we observe that there is no significant leakage with $1 \times 10^6$ traces. An alternative would be to set the coordinates of $R_n$ to zero before setting $R_n$ to $R_2$. Algorithm 9 shows the same arguments applied to Algorithm 7. However there is no need to randomize any points during the loops of scalar multiplication. If the redundant representation of the point assigned to $R_0$ and $R_1$ is randomized separately to that applied to $U_0$ an overwrite with a Hamming distance of zero cannot occur.

Algorithm 8: Montgomery Ladder with XOR-Split Scalar on EC.

```
Input: $E, \mathcal{F}_q, P \in E, n$-bit integers
$A = \sum_{i=0}^{n-1} a_i 2^i, B = \sum_{i=0}^{n-1} b_i 2^i$
Output: $Q = [\kappa]P$ where $\kappa = A \oplus B$
Uses: $C = \sum_{i=0}^{n-1} c_i 2^i$
1 $R_0 \leftarrow P; R_1 \leftarrow P; R_2 \leftarrow P$
2 $C \leftarrow B \oplus \left[ \frac{B}{2} \right]$;
3 $b' \leftarrow b_{n-1}$;
4 $R_{-b'} \leftarrow 2P$;
5 for $i = n - 2$ down to 0 do
6 $R_2 \leftarrow R_0 + R_{-a_i};$
7 $R_0 \leftarrow 2R_{0(\oplus \gamma_i)}$;
8 $R_{-a_i} \leftarrow R_2$;
9 end
10 return $R_{b_0}$
```

Algorithm 9: Montgomery Ladder with XOR-Split Scalar II on EC.

```
Input: $E, \mathcal{F}_q, P \in E, n$-bit integers
$A = \sum_{i=0}^{n-1} a_i 2^i, B = \sum_{i=0}^{n-1} b_i 2^i$
Output: $Q = [\kappa]P$ where $\kappa = A \oplus B$
Uses: $C = \sum_{i=0}^{n-1} c_i 2^i$ and $D = \sum_{i=0}^{n-1} d_i 2^i$
1 $R_0 \leftarrow P; R_1 \leftarrow P$
2 $U_0 \leftarrow P; U_1 \leftarrow P$
3 $C \leftarrow B \oplus \left[ \frac{B}{2} \right]; D \leftarrow C \oplus A$;
4 $b' \leftarrow b_{n-1}; R_{-b'} \leftarrow 2P$;
5 for $i = n - 2$ down to 0 do
6 $R_0 \leftarrow R_0 + R_{\gamma_i}$;
7 $R_1 \leftarrow R_0 + U_{b_i}$;
8 $b' \leftarrow b_i$;
9 end
10 return $R_{b_0}$
```

6 CONCLUSIONS

In this paper, we show how an exponent can be split into two shares, where the exponent is the XOR sum of the two shares and the cost is typically an extra register and some register copies per bit. A significant advantage over previously proposed exponent splitting methods, which can have a prohibitive impact on performance (Clavier and Joye, 2001). Our method can also be applied to groups whose order contains long runs of bits set to 0 or 1 without any penalty on performance or security. Indeed, one does not need to know the order of the group; a significant advantage if, for example, one wished to implement RSA
without using the Chinese remainder theorem.

We show that our algorithms are secure using formal methods, MI-based evaluation and TVLA on an implementation of Boolean exponent splitting. We note that our method does not prevent an attacker from using the intermediate states generated by the algorithms as a means of attack. However, inexpensive solutions such as randomizing projective points (Win et al., 1998) or Ebeid and Lambert’s blinding method for RSA (Ebeid and Lambert, 2010) can be combined with our method to provide a high level of side-channel resistance.

The algorithms presented above will be more efficient than adding a multiple of the group order to the exponent, since the bit length of the exponent is not increased. Moreover, the resistance to collision attacks is superior, since one would need to conduct several attacks to derive each share and reconstruct the exponent. Where one is adding a random multiple of the exponent any bits recovered directly relate to bits of the exponent used. It has not yet been shown that one can derive an exponent from gaining partial information on a series of blinded exponents, but significant advances have been made (Schindler and Itoh, 2011; Joye and Lepoint, 2012; Schindler, 2014; Schindler and Wiemers, 2014).

ACKNOWLEDGEMENTS

The authors would like to thank Lauren De Meyer and Michael Hamburg for their helpful comments.

REFERENCES


